选择排序:

算法步骤:对于下标为i的元素来说需要从i ~ n找到第1小的元素，最后和下标为i的元素进行交换
稳定性：由于存在交换所以选择排序一定是不稳定的 举例 4 4 2
时间复杂度: o(n^2)
空间复杂度: o(n)

```
void selection_sort(int* a, int n) {
  for (int i = 1; i < n; ++i) {
    int ith = i;
    for (int j = i + 1; j <= n; ++j) {
      if (a[j] < a[ith]) {
        ith = j;
      }
    }
    std::swap(a[i], a[ith]);
  }
}
```

思考:能不能在交换途中求出逆序对个数?

冒泡排序:

算法步骤:每一次遍历全部未排序的元素，前后比较交换
稳定性：稳定(由于每次是前后元素比较交换)
空间复杂度: o(n)
时间复杂度: o(n^2) 最优复杂度 on 平均复杂度 o(n^2)

```
// C++ Version
// 假设数组的大小是n+1，冒泡排序从数组下标1开始
void bubble_sort(int *a, int n) {
  bool flag = true;
  while (flag) {
    flag = false;
    for (int i = 1; i < n; ++i) {
      if (a[i] > a[i + 1]) {
        flag = true;
        a[i] ^= a[i + 1];
        a[i + 1] ^= a[i];
        a[i] ^= a[i + 1];
      }
    }
  }
}
```

思考:能不能在交换途中求出逆序对个数?

```
// C++ Version
// 假设数组的大小是n+1，冒泡排序从数组下标1开始
void bubble_sort(int *a, int n) {
  bool flag = true;
  while (flag) {
    flag = false;
    for (int i = 1; i < n; ++i) {
      if (a[i] > a[i + 1]) {
        flag = true;
        a[i] ^= a[i + 1];
        a[i + 1] ^= a[i];
        a[i] ^= a[i + 1];
        sum++;
      }
    }
  }
}
```

插入排序:

将排列序列分为未排序序列和已排序序列每次从未排序序列中选一个插入到已排序序列
算法步骤:每次从i遍历到1 若 a[x] > a[j]则a[j]移到a[j + 1]
稳定性: 稳定 ， 和冒泡排序类似
时间复杂度:最优复杂度 on 最坏复杂度 o(n^2) 平均复杂度o(n^2)
空间复杂度: on

```
// C++ Version
void insertion_sort(int* a, int n) {
  // 对 a[1],a[2],...,a[n] 进行插入排序
  for (int i = 2; i <= n; ++i) {
    int key = a[i];
    int j = i - 1;
    while (j > 0 && a[j] > key) {
      a[j + 1] = a[j];
      --j;
    }
    a[j + 1] = key;
  }
}
```

思考:能不能在交换途中求出逆序对个数?

```
// C++ Version
void insertion_sort(int* a, int n) {
  // 对 a[1],a[2],...,a[n] 进行插入排序
  for (int i = 2; i <= n; ++i) {
    int key = a[i];
    int j = i - 1;
    while (j > 0 && a[j] > key) {
```

```
      a[j + 1] = a[j];
      --j;
      sum++;
    }
    a[j + 1] = key;
  }
}
```

计数排序:

计数排序：一种线性的排序算法
算法步骤:对原数组a来说建立一个新的数组b，b[i]表示的是 a数组中元素大小为i的个数，随后再创建一个数组c，c[i]表示的是 b[1] ~ b[i]的和。那么c[i] - c[i - 1]就是 a数组中元素大小为i的个数，并且我们知道 这个数排序后的下标在c[i - 1] + 1 ~ c[i]
稳定性：稳定
时间复杂度: o(n + w)
空间复杂度 o(n + w)

```
void counting_sort() {
  memset(cnt, 0, sizeof(cnt));
  for (int i = 1; i <= n; ++i) ++cnt[a[i]];
  for (int i = 1; i <= w; ++i) cnt[i] += cnt[i - 1];
  for (int i = n; i >= 1; --i) b[cnt[a[i]]--] = a[i];
}
```

练习:

1. U248358 输入一个数n 和 n个整数,$a_1, a_2 \ldots, a_n$，将这些整数按照从小到大排序 n <= 5e6,a_i <= 1e7

思考:能不能在交换途中求出逆序对个数?

基数排序

定义：基数排序是基于位来选择其他排序算法进行排序的
算法步骤:分别将位数设置第一个关键字，第二关键字..第k关键词，然后先排序第一关键字再排序第二关键字....
空间复杂度 on
时间复杂度 o(k/x * (n + pow(10,x)) k是位数，x是几位为一组
稳定性:稳定

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int maxn = 5e6 + 5;
int n, a[maxn], maxx, k;
vector<int> v[maxn], v2[maxn];
signed main(){
```

```
    cin >> n;
    for (int i = 1; i <= n; i++){
        cin >> a[i];
        maxx = max(maxx, a[i]);
    }
    while (maxx){
        k++;
        maxx /= 10;
    }
    for (int i = 1; i <= n; i++){
        for (int j = k; j >= 1; j--){
            v[i].push_back((a[i] / (int)pow(10, j - 1)) % 10);
        }
    }
    for (int j = k - 1; j >= 0; j--){
        int cnt[10] = {};
        for (int i = 1; i <= n; i++){
            cnt[v[i][j]]++;
        }
        for (int i = 1; i <= 10; i++){
            cnt[i] += cnt[i - 1];
        }
        for (int i = n; i >= 1; i--){
            v2[cnt[v[i][j]]--] = v[i];
        }
        for (int i = 1; i <= n; i++){
            v[i] = v2[i];
        }
    }
    for (int i = 1; i <= n; i++){
        for (int j = 0; j < v[i].size(); j++){
            cout << v[i][j];
        }
        cout << endl;
    }
    return 0;
}
```

思考:能不能在交换途中求出逆序对个数?

练习:

1. U248368 输入一个数n 和 n个整数,$a_1, a_2 \ldots, a_n$，将这些整数按照从小到大排序 n <= 5e6,a_i <= 1e7

2. 51nod3236

快速排序

快速排序：一个基于分治思想的排序
算法步骤:在当前序列中确定一个基准数，使得当前序列相对有序。
稳定性:不稳定
时间复杂度:o(nlogn) 最坏复杂度o(n^2) 最坏复杂度举例: 1 1 1 1 1 1 1 1
空间复杂度:on

```
struct Range {
  int start, end;

  Range(int s = 0, int e = 0) { start = s, end = e; }
};
template <typename T>
void quick_sort(T arr[], const int len) {
  if (len <= 0) return;
  Range r[len];
  int p = 0;
  r[p++] = Range(0, len - 1);
  while (p) {
    Range range = r[--p];
    if (range.start >= range.end) continue;
    T mid = arr[range.end];
    int left = range.start, right = range.end - 1;
    while (left < right) {
      while (arr[left] < mid && left < right) left++;
      while (arr[right] >= mid && left < right) right--;
      std::swap(arr[left], arr[right]);
    }
    if (arr[left] >= arr[range.end])
      std::swap(arr[left], arr[range.end]);
    else
      left++;
    r[p++] = Range(range.start, left - 1);
    r[p++] = Range(left + 1, range.end);
  }
}
```

快速排序优化:
1.通过 三数取中（即选取第一个、最后一个以及中间的元素中的中位数） 的方法来选择两个子序列的分界元素（即比较基准）。这样可以避免极端数据（如升序序列或降序序列）带来的退化；
2.当序列较短时，使用 插入排序 的效率更高；
3.每趟排序后，将与分界元素相等的元素聚集在分界元素周围，这样可以避免极端数据（如序列中大部分元素都相等）带来的退化。

```
template <typename T>
// arr 为需要被排序的数组，len 为数组长度
void quick_sort(T arr[], const int len) {
  if (len <= 1) return;
  // 随机选择基准（pivot）
  const T pivot = arr[rand() % len];
  // i：当前操作的元素
  // j：第一个等于 pivot 的元素
  // k：第一个大于 pivot 的元素
  int i = 0, j = 0, k = len;
  // 完成一趟三路快排，将序列分为：
  // 小于 pivot 的元素 | 等于 pivot 的元素 | 大于 pivot 的元素
```

```
  while (i < k) {
    if (arr[i] < pivot)
      swap(arr[i++], arr[j++]);
    else if (pivot < arr[i])
      swap(arr[i], arr[--k]);
    else
      i++;
  }
  // 递归完成对于两个子序列的快速排序
  quick_sort(arr, j);
  quick_sort(arr + k, len - k);
}
```

思考:能不能在交换途中求出逆序对个数?

求第k大值

基于快速排序的思想
算法步骤：每次确定一个基准点，小于基准点的放在一个数组里，元素个数为x，等于基准点的放在一个数组里，元素个数为y，大于基准点的放在一个基准点里，元素个数为z，若k大于 x + y那么 一定在x + y + 1 ~ z里面，否则一定在1 ~ x + y里面
算法复杂度: 最优复杂度on，最坏复杂度n^2
空间复杂度: onlogn

```
template <typename T>
T find_kth_element(T arr[], int rk, const int len) {
  if (len <= 1) return arr[0];
  // 随机选择基准（pivot）
  const T pivot = arr[rand() % len];
  // i：当前操作的元素
  // j：第一个等于 pivot 的元素
  // k：第一个大于 pivot 的元素
  int i = 0, j = 0, k = len;
  // 完成一趟三路快排，将序列分为：
  // 小于 pivot 的元素 | 等于 pivot 的元素 | 大于 pivot 的元素
  while (i < k) {
    if (arr[i] < pivot)
      swap(arr[i++], arr[j++]);
    else if (pivot < arr[i])
      swap(arr[i], arr[--k]);
    else
      i++;
  }
  if (rk < j) return find_kth_element(arr, rk, j);
  else if (rk >= k)
    return find_kth_element(arr + k, rk - k, len - k);
  return pivot;
}
```

练习

1. P1177 利用快速排序算法将读入的 NN 个数从小到大排序后输出 N≤10^5

2. P1138 现有 n个正整数，要求出这 n个正整数中的第 k个最小整数（相同大小的整数只计算一次）。 n≤10000，k ≤1000，正整数均小于 30000。

3. U248371 输入一个数 n，然后n个整数，然后一个整数k表示第k大 n <= 5e6,a_i <= 2e9,k <= n。

归并排序:

归并排序：基于分治的思想
算法步骤:对于排序 1 ~ n我们可以先排序 1 ~ n / 2,n / 2 + 1 ~ n以此类推再合并
稳定性:稳定
复杂度：onlogn
空间复杂度 : on

```
void merge(int l, int r) {
  if (r - l <= 1) return;
  int mid = l + ((r - l) >> 1);
  merge(l, mid), merge(mid, r);
  for (int i = l, j = mid, k = l; k < r; ++k) {
    if (j == r || (i < mid && a[i] <= a[j]))
      tmp[k] = a[i++];
    else
      tmp[k] = a[j++];
  }
  for (int i = l; i < r; ++i) a[i] = tmp[i];
}
```

思考:能不能在交换途中求出逆序对个数?

```
void merge(int l, int r) {
  if (r - l <= 1) return;
  int mid = l + ((r - l) >> 1);
  merge(l, mid), merge(mid, r);
  for (int i = l, j = mid, k = l; k < r; ++k) {
    if (j == r || (i < mid && a[i] <= a[j]))
      tmp[k] = a[i++];
    else{
         cnt += mid - i;
        tmp[k] = a[j++];
    }
  }
```

```
  for (int i = l; i < r; ++i) a[i] = tmp[i];
}
```

桶排序

桶排序（英文：Bucket sort）是排序算法的一种，适用于待排序数据值域较大但分布比较均匀的情况，同样需要内置排序。
算法步骤:设置一个定量的数组当作空桶；遍历序列，并将元素一个个放到对应的桶中；对每个不是空的桶进行排序；从不是空的桶里把元素再放回原来的序列中。
稳定性:若内置函数是稳定的就是稳定的，一般使用插入排序
空间复杂度: on
时间复杂度: o(n + n^2/k + k)

```
// C++ Version
const int N = 100010;

int n, w, a[N];
vector<int> bucket[N];

void insertion_sort(vector<int>& A) {
  for (int i = 1; i < A.size(); ++i) {
    int key = A[i];
    int j = i - 1;
    while (j >= 0 && A[j] > key) {
      A[j + 1] = A[j];
      --j;
    }
    A[j + 1] = key;
  }
}

void bucket_sort() {
  int bucket_size = w / n + 1;
  for (int i = 0; i < n; ++i) {
    bucket[i].clear();
  }
  for (int i = 1; i <= n; ++i) {
    bucket[a[i] / bucket_size].push_back(a[i]);
  }
  int p = 0;
  for (int i = 0; i < n; ++i) {
    insertion_sort(bucket[i]);
    for (int j = 0; j < bucket[i].size(); ++j) {
      a[++p] = bucket[i][j];
    }
  }
}
```

思考:能不能在交换途中求出逆序对个数?

希尔排序

希尔排序（英语：Shell sort），也称为缩小增量排序法，是 插入排序 的一种改进版本
由于插入排序的复杂度取决于序列的有序程度，所以希尔排序是先使得它相对有序，再进行最后的排序
算法步骤:排序对不相邻的记录进行比较和移动：

1. 将待排序序列分为若干子序列（每个子序列的元素在原始数组中间距相同）；
2. 对这些子序列进行插入排序；
3. 减小每个子序列中元素之间的间距，重复上述过程直至间距减少为 1。

稳定性:不稳定
算法复杂度: $o(n^{1.3)\sim o(n}2)$

```
template <typename T>
void shell_sort(T array[], int length) {
  int h = 1;
  while (h < length / 3) {
    h = 3 * h + 1;
  }
  while (h >= 1) {
    for (int i = h; i < length; i++) {
      for (int j = i; j >= h && array[j] < array[j - h]; j -= h) {
        std::swap(array[j], array[j - h]);
      }
    }
    h = h / 3;
  }
}
```

思考:能不能在交换途中求出逆序对个数?